EXHIBIT 1

Annotated XY Token Smart Contract Code

Note: Below is the XY Token Smart Contract code deployed to Ethereum Mainnet on December 6th, 2021. Descriptive annotations are provided in *red italics* directly preceding the code in **bold**.

+ +

```
pragma solidity 0.8.2;

abstract contract Context {
   function _msgSender() internal view virtual returns (address) {
      return msg.sender;
   }

   function _msgData() internal view virtual returns (bytes calldata) {
      return msg.data;
   }
}

// Part: OpenZeppelin/openzeppelin-contracts@4.3.2/IERC20

 * Interface of the ERC20 standard as defined in the EIP.
interface IERC20 {

    *   Returns the amount of tokens in existence.
   function totalSupply() external view returns (uint256);

    *   Returns the amount of tokens owned by `account`.
   function balanceOf(address account) external view returns (uint256);

    *   Moves `amount` tokens from the caller's account to `recipient`.
  function transfer(address recipient, uint256 amount) external returns (bool);

    * Returns the remaining number of tokens that `spender` will be allowed to spend on behalf
of `owner` through {transferFrom}.
    * This is zero by default.
    * This value changes when {approve} or {transferFrom} are called.
   function allowance(address owner, address spender) external view returns (uint256);

    *   Sets `amount` as the allowance of `spender` over the caller's tokens.
   function approve(address spender, uint256 amount) external returns (bool);

    *   Moves `amount` tokens from `sender` to `recipient` using the allowance mechanism.
    *  `amount` is then deducted from the caller's allowance.
   function transferFrom(
      address sender,
      address recipient,
      uint256 amount
   ) external returns (bool);

    *  Emitted when `value` tokens are moved from one account (`from`) to another (`to`).
    * Note that `value` may be zero.
   event Transfer(address indexed from, address indexed to, uint256 value);

    *  Emitted when the allowance of a `spender` for an `owner` is set by a call to {approve}.
    * `value` is the new allowance.
   event Approval(address indexed owner, address indexed spender, uint256 value);
}

// Part: OpenZeppelin/openzeppelin-contracts@4.3.2/IERC20Metadata

 *  Interface for the optional metadata functions from the ERC20 standard.
interface IERC20Metadata is IERC20 {

    *  Returns the name of the token.
   function name() external view returns (string memory);

    *  Returns the symbol of the token.
   function symbol() external view returns (string memory);

    *  Returns the decimals places of the token.
   function decimals() external view returns (uint8);
}

// Part: OpenZeppelin/openzeppelin-contracts@4.3.2/Ownable

 *  Contract module which provides a basic access control mechanism, where there is an
account (an owner) that can be granted exclusive access to specific functions.
 * By default, the owner account will be the one that deploys the contract.
 * This can later be changed with {transferOwnership}.
abstract contract Ownable is Context {
   address private _owner;

   event OwnershipTransferred(address indexed previousOwner, address indexed
newOwner);

    *  Initializes the contract setting the deployer as the initial owner.
   constructor() {
      _setOwner(_msgSender());
   }

    *   Returns the address of the current owner.
    *  Current owner's address is 0x0000000000000000000000000000000000000000 address,
a burn address
   function owner() public view virtual returns (address) {
      return _owner;
   }

    *   Throws if called by any account other than the owner.
   modifier onlyOwner() {
      require(owner() == _msgSender(), "Ownable: caller is not the owner");
      _;
   }

    * Leaves the contract without owner.
    * It will not be possible to call`onlyOwner` functions anymore.
    * Can only be called by the current owner.
   function renounceOwnership() public virtual onlyOwner {
      _setOwner(address(0));
   }

    *   Transfers ownership of the contract to a new account (`newOwner`).
    * Can only be called by the current owner.
   function transferOwnership(address newOwner) public virtual onlyOwner {
      require(newOwner != address(0), "Ownable: new owner is the zero address");
      _setOwner(newOwner);
   }

   function _setOwner(address newOwner) private {
      address oldOwner = _owner;
      _owner = newOwner;
      emit OwnershipTransferred(oldOwner, newOwner);
   }
}

// Part: OpenZeppelin/openzeppelin-contracts@4.3.2/ERC20

 *   Implementation of the {IERC20} interface.
contract ERC20 is Context, IERC20, IERC20Metadata {
   mapping(address => uint256) private _balances;
   mapping(address => mapping(address => uint256)) private _allowances;

   uint256 private _totalSupply;

   string private _name;
   string private _symbol;

    *   Sets the values for {name} and {symbol}.
    * All two of these values are immutable: they can only be set once during construction.
   constructor(string memory name_, string memory symbol_) {
      _name = name_;
      _symbol = symbol_;
   }

    *   Returns the name of the token.
   function name() public view virtual override returns (string memory) {
      return _name;
   }

    *   Returns the symbol of the token, usually a shorter version of the name.
   function symbol() public view virtual override returns (string memory) {
      return _symbol;
   }

    *   Returns the number of decimals used to get its user representation.
      function decimals() public view virtual override returns (uint8) {
      return 18;
   }

      *   See {IERC20-totalSupply}.
   function totalSupply() public view virtual override returns (uint256) {
      return _totalSupply;
   }

    *   See {IERC20-balanceOf}.
   function balanceOf(address account) public view virtual override returns (uint256) {
      return _balances[account];
   }

    *   See {IERC20-transfer}.
    * Requirements:
    * - `recipient` cannot be the zero address.
    * - the caller must have a balance of at least `amount`.
```

```
   function transfer(address recipient, uint256 amount) public virtual override returns
(bool) {
      _transfer(_msgSender(), recipient, amount);
      return true;
   }

    *   See {IERC20-allowance}.
   function allowance(address owner, address spender) public view virtual override
returns (uint256) {
      return _allowances[owner][spender];
   }

    *   See {IERC20-approve}.
    * Requirements:
    * - `spender` cannot be the zero address.
   function approve(address spender, uint256 amount) public virtual override returns
(bool) {
      _approve(_msgSender(), spender, amount);
      return true;
   }

    *   See {IERC20-transferFrom}.
    * Emits an {Approval} event indicating the updated allowance.
    * Requirements:
    * - `sender` and `recipient` cannot be the zero address.
    * - `sender` must have a balance of at least `amount`.
    * - the caller must have allowance for ``sender``'s tokens of at least `amount`.
   function transferFrom(
      address sender,
      address recipient,
      uint256 amount
   ) public virtual override returns (bool) {
      _transfer(sender, recipient, amount);

      uint256 currentAllowance = _allowances[sender][_msgSender()];
      require(currentAllowance >= amount, "ERC20: transfer amount exceeds
allowance");
      unchecked {
         _approve(sender, _msgSender(), currentAllowance - amount);
      }

      return true;
   }
```

```
    *   Atomically increases the allowance granted to `spender` by the caller.
    * Emits an {Approval} event indicating the updated allowance.
    * Requirements:
    * - `spender` cannot be the zero address.
   function increaseAllowance(address spender, uint256 addedValue) public virtual
returns (bool) {
       _approve(_msgSender(), spender, _allowances[_msgSender()][spender] +
addedValue);
       return true;
   }

    *   Atomically decreases the allowance granted to `spender` by the caller.
    * Emits an {Approval} event indicating the updated allowance.
    * Requirements:
    * - `spender` cannot be the zero address.
    * - `spender` must have allowance for the caller of at least `subtractedValue`.
   function decreaseAllowance(address spender, uint256 subtractedValue) public virtual
returns (bool) {
       uint256 currentAllowance = _allowances[_msgSender()][spender];
       require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below
zero");
       unchecked {
           _approve(_msgSender(), spender, currentAllowance - subtractedValue);
       }

       return true;
   }

    *   Moves `amount` of tokens from `sender` to `recipient`.
    * This internal function is equivalent to {transfer}
    * Emits a {Transfer} event.
    * Requirements:
    * - `sender` cannot be the zero address.
    * - `recipient` cannot be the zero address.
    * - `sender` must have a balance of at least `amount`.
   function _transfer(
       address sender,
       address recipient,
       uint256 amount
   ) internal virtual {
       require(sender != address(0), "ERC20: transfer from the zero address");
       require(recipient != address(0), "ERC20: transfer to the zero address");

       _beforeTokenTransfer(sender, recipient, amount);
       uint256 senderBalance = _balances[sender];
       require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
       unchecked {
           _balances[sender] = senderBalance - amount;
       }
       _balances[recipient] += amount;

       emit Transfer(sender, recipient, amount);

       _afterTokenTransfer(sender, recipient, amount);
   }

   /**   Creates `amount` tokens and assigns them to `account`, increasing the total supply.
    * Emits a {Transfer} event with `from` set to the zero address.
    * Requirements:
    * - `account` cannot be the zero address.
   function _mint(address account, uint256 amount) internal virtual {
       require(account != address(0), "ERC20: mint to the zero address");

       _beforeTokenTransfer(address(0), account, amount);

       _totalSupply += amount;
       _balances[account] += amount;
       emit Transfer(address(0), account, amount);

       _afterTokenTransfer(address(0), account, amount);
   }

    *   Destroys `amount` tokens from `account`, reducing the total supply.
    * Emits a {Transfer} event with `to` set to the zero address.
    * Requirements:
    * - `account` cannot be the zero address.
    * - `account` must have at least `amount` tokens.
   function _burn(address account, uint256 amount) internal virtual {
       require(account != address(0), "ERC20: burn from the zero address");

       _beforeTokenTransfer(account, address(0), amount);

       uint256 accountBalance = _balances[account];
       require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
       unchecked {
           _balances[account] = accountBalance - amount;
       }
```

```
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }

     *   Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     * This internal function is equivalent to `approve`
     * Emits an {Approval} event.
     * Requirements:
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

     *   Hook that is called before any transfer of tokens. This includes minting and burning.
     * Calling conditions:
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens will be transferred to
 `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
      function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}

     *   Hook that is called after any transfer of tokens. This includes minting and burning.
     * Calling conditions:
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens has been
 transferred to `to`.
     * - when `from` is zero, `amount` tokens have been minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens have been burned.
     * - `from` and `to` are never both zero.
     function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
}

// File: XYToken.sol

/// @title XYToken is the XY Finance governance token
contract XYToken is ERC20, Ownable {

    ///   This contract should be deployed on all periphery chains.
    ///   - On Ethereum, `amount` is set to `100,000,000 * 1e18` and `renounceOwnership` should
be called right after the contract is deployed, to make sure the cap is `100,000,000 * 1e18`.
    ///   - On other chains, `amount` is set to `0`. The contract is served as a XY Token bridge
through mint-and-burn.
    /// @param name XY Token name
    /// @param symbol XY Token symbol
    /// @param vault Address where initial `amount` XY Token is sent
    /// @param amount Amount of XY Token is minted when the contract is deployed
    constructor(string memory name, string memory symbol, address vault, uint256
amount) ERC20(name, symbol) {
        _mint(vault, amount);
    }

    mapping (address => bool) public isMinter;

    modifier onlyMinter {
        require(isMinter[msg.sender], "ERR_NOT_MINTER");
        _;
    }

Gives “address minter” minting privileges
    function setMinter(address minter, bool _isMinter) external onlyOwner {
        isMinter[minter] = _isMinter;

        emit SetMinter(minter, _isMinter);
    }

Function mints amount to associated account
    function mint(address account, uint256 amount) external onlyMinter {
        _mint(account, amount);
    }
```

Function will burn amount from account balance

```
    function burn(uint256 amount) external {
        _burn(msg.sender, amount);
    }

    event SetMinter(address minter, bool isMinter);
}
```